Computer Science Department

Indiana University

Lindley Hall 101

Bloomington, Indiana 47401

Key Words and Phrases:

structured programming, control structures, looping, while statement, until statement, exit statement, flowchart, ease of programming.

CR Categories: 4.22; 4.42; 5.24

TECHNICAL REPORT NO. 10

A CASE FOR WHILE-UNTIL

DANIEL P. FRIEDMAN

STUART C. SHAPIRO

APRIL 5, 1974

Appeared in SIGPLAN Notices 9, 7 (July, 1974), 7-14.

A CASE FOR WHILE-UNTIL

Daniel P. Friedman
Stuart C. Shapiro
Indiana University

Abstract

A new control structure construct, the while-until, is introduced as a syntactic combination of the while and the until. Examples are shown indicating that use of the while-until can lead to structured programs that are conceptually more manageable than those attainable without it. The while-until statement is then extended to a value-returning expression which is shown to be more powerful than either the while or the until.

A major suggestion of structured programming is to employ looping control structures in order to break the program down into conceptually manageable units. The purpose of this paper is to propose an additional looping control structure construct (the while-until) that, in certain instances, yields program loops that are closer to the conceptual organization of the segment than is possible with the existing constructs. The while-until as a statement will be shown to be equivalent to the existing looping control structures. The while-until as a value (Boolean) returning expression will be shown to be a more powerful control structure than the while or until structures discussed by Dijkstra [1].

The existing constructs that we are concerned with are while β repeat s

and

repeat s until β

Dijkstra [1] presents these graphically as in Figures 1 and 2.

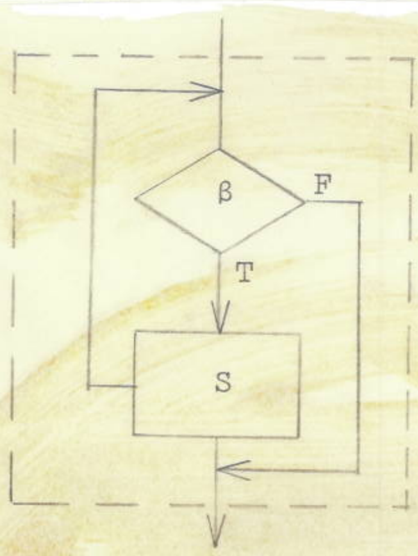

Fig. 1 while β repeat s

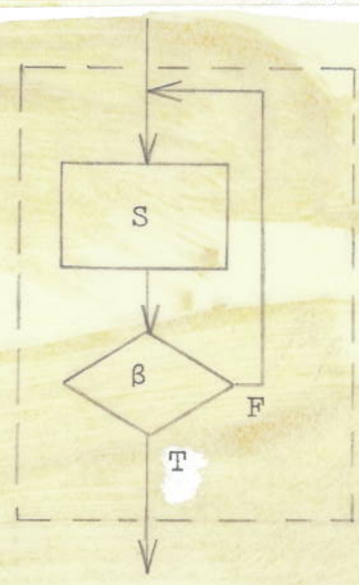

Fig. 2 repeat s until β

The syntactic construct we are proposing is

$$\underline{\text{while}}\ \beta_1\ \underline{\text{repeat}}\ s\ \underline{\text{until}}\ \beta_2$$

which is presented graphically in Figure 3.

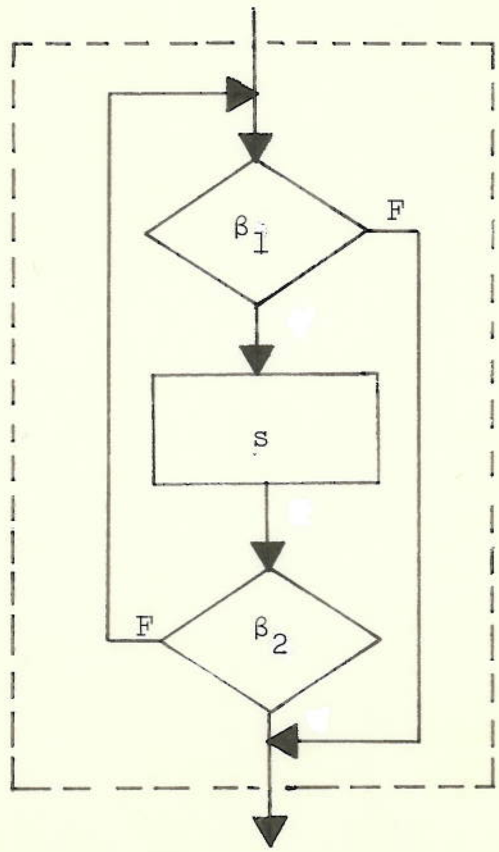

Fig. 3 $\underline{\text{while}}\ \beta_1\ \underline{\text{repeat}}\ s\ \underline{\text{until}}\ \beta_2$

The while-until does not involve nesting, but is some other combination [2] of the features of the while and the until loops. The while-until may be replaced by the until or the while as the only looping structure since

$$\underline{\text{while}}\ \beta_1\ \underline{\text{repeat}}\ s\ \underline{\text{until}}\ \beta_2$$

is equivalent to

$$\underline{\text{if}}\ \beta_1\ \underline{\text{then}}\ \underline{\text{repeat}}\ s\ \underline{\text{until}}\ \underline{\text{if}}(\beta_2\ \underline{\text{then}}\ \underline{\text{true}}\ \underline{\text{else}}\ \neg\beta_1)$$

and also to

$$\underline{\text{if}}\ \beta_1\ \underline{\text{then}}\ \underline{\text{begin}}\ s;\ \underline{\text{while}}\ \underline{\text{if}}(\beta_2\ \underline{\text{then}}\ \underline{\text{false}}\ \underline{\text{else}}\ \beta_1)\ \underline{\text{repeat}}\ s\ \underline{\text{end}}$$

If escape (or break or exit) were employed, another equivalent form is

```
A:  while β1 repeat
        begin s; if β2 then escape A end
    end A
```

In those cases where Figure 3 is the desired control structure it appears that the while-until yields clearer, more understandable code than any of the above alternatives. We can paraphrase the semantic content of the while-until as: "while it is possible to try, keep trying until you succeed." The while and the until loops can be defined in terms of the while-until in the following way:

while β repeat s $=_{def}$ while β repeat s until false

and

repeat s until β $=_{def}$ while true repeat s until β

The while-until is a natural control structure for searching, since every search terminates either by finding the desired element or by determining that it is not present. As an example, we show its use for a binary search:

```
comment Find item A in table T[1:N] ;
low :=0;
high := N + 1;
while low < high - 1 repeat
     try := (low + high) / 2;
     if T[try] < A then low := try else high := try
until T[try] = A;
```

An appropriate application for the while-until occurs whenever a loop includes two operations, one of which requiring a test prior to its execution and the other requiring a test which can only be performed after its execution. An example of this is: copy a file up to and including the end-of-file mark onto an output file, however, nothing may be written on the output file unless there is enough space for a record.

```
comment Copy file INPUT onto the file OUTPUT;
while Spaceleft(OUTPUT) repeat
     Inbuffer(INPUT, b);
     Outbuffer(OUTPUT, b)
until Eof(INPUT);
```

In languages in which statements are expressions having values, for example LISP [3], ALGOL 68 [4] and BLISS [5], the while-until can be assigned a value in an especially useful way. We define the value of the while-until expression to be the value of the last evaluated Boolean. That is, the value of

while β_1 repeat s until β_2

is false if and only if the loop terminates due to the evaluation of β_1 (see Figure 4). A non-Boolean value could be returned on certain termination conditions (e.g. exit in BLISS or predicates in LISP).

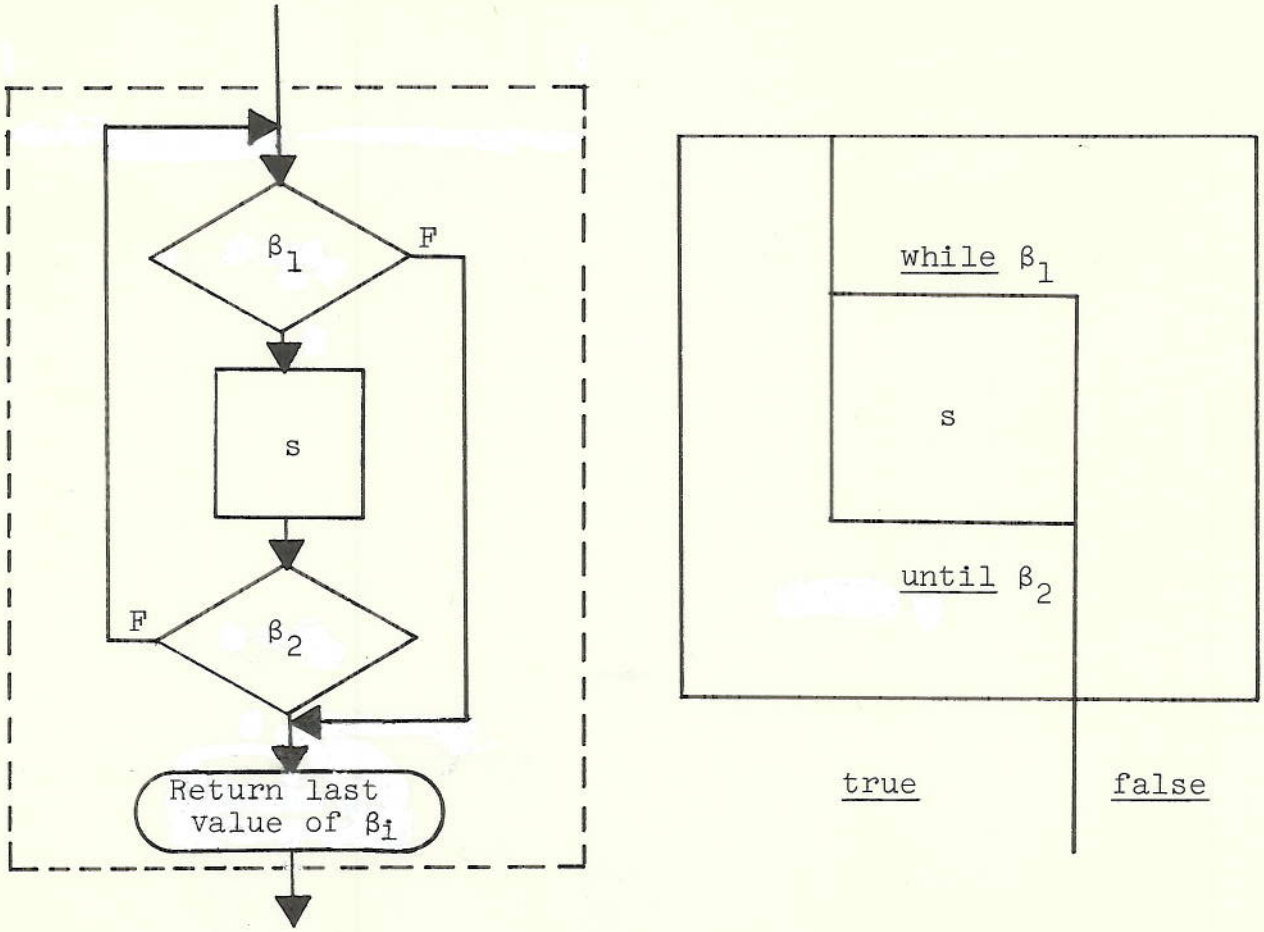

Fig. 4 value returning while-until, illustrated as standard flowcharts and a proposal for structured flowchart [6].

There are two ways in which the loop may be terminated: the programmer will want to ascertain which of the two Booleans caused termination. This is precisely the information provided by the value of the while-until.

Using the value of the while-until, we may easily incorporate the above search routine into an insertion.

```
comment T[1:M] is a table containing  N < M  active elements.
    Insert A in T if it is not already present;
low := 0;
high := N + 1;
if¬(while low < high - 1 repeat
    try := (low + high) / 2
    if T[try] < A then low := try else high := try
  until T[try] = A)
 then Insertafter(A,T,low);
```

The previously presented copy routine can be incorporated into an algorithm that uses up to N output files, depending on the length of the input file:

```
comment Place one copy of file INPUT onto OUTPUT[1:N] as needed;
i := 0
while i < N repeat
            if i > 0 then Close(OUTPUT[i]);
            i := i + 1
            Open(OUTPUT[i])
until (while Spaceleft(Output[i]) repeat
            Inbuffer(INPUT, b)
            Outbuffer(OUTPUT[i], b)
            until Eof(INPUT));
Close(OUTPUT[i])
```

Earlier, we showed that the while-until statement is definable in terms of just the while or just the until. This is not true, however, for the while-until expression. Peterson, Kasami, and Tokura [7], p. 506, have shown that "There exist flowcharts that cannot be translated into [if and until] programs with single-level exits, even if node splitting is allowed." Their example of such a flowchart is shown in Figure 5. The following program using value-returning while-until and if expression is a translation of this flowchart.

```
S; while (if A then
                (if (while true repeat a1 until true)
                  then (if (while B repeat b1
                          until (if C then true
                                 else ¬(while true repeat c2 until true)))
                      then (while true repeat c1 until true)
                      else ¬(while true repeat b2 until true))
                  else false)
          else (while true repeat a2 until true))
   repeat until (if D then ¬(while true repeat d1 until true)
                 else (while true repeat d2 until true)); T
```

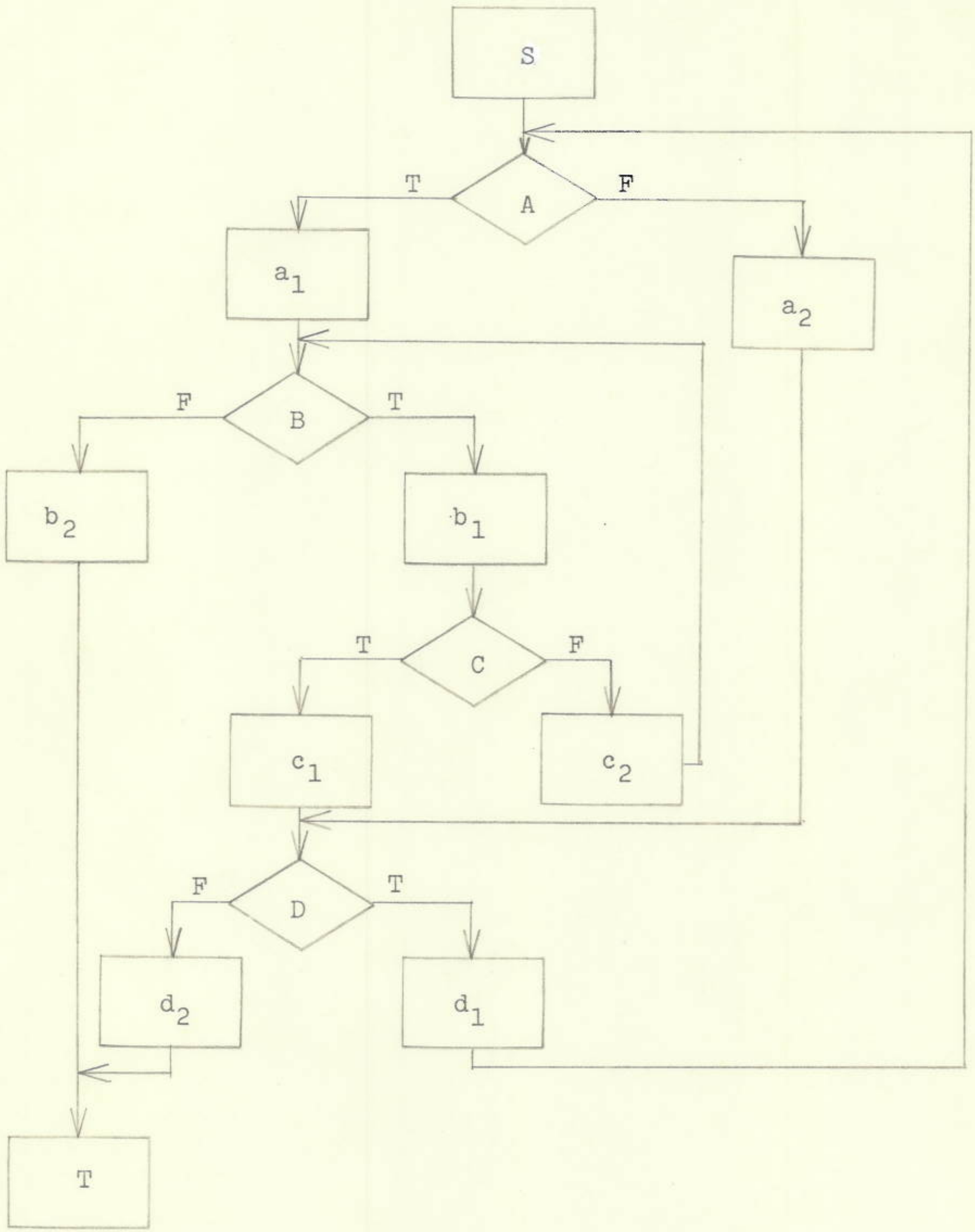

Fig. 5 Flowchart from Peterson, Kasami, and Tokura [7].

Similarly, Ashcroft and Manna [8] have exhibited a flowchart, shown in Figure 6, which cannot be translated into an if and while program. The following while-until program, due to M. Wand and D. Wise, is a translation of this flowchart.

```
if (while (if (while P repeat h until false) then true
                                             else Q)
     repeat h
     until ¬(while (if (while Q repeat g until false) then true
                                                      else P)
         repeat g until true))
then g else h
```

We have introduced the while-until as an additional control structure for structured programming. We have demonstrated cases in which use of the while-until results in more readable programs and allows programmers to program closer to the way they think. Although the while-until statement can be defined in terms of the while or the until, we have shown that the while-until expression is a stronger control structure than either.

-8-

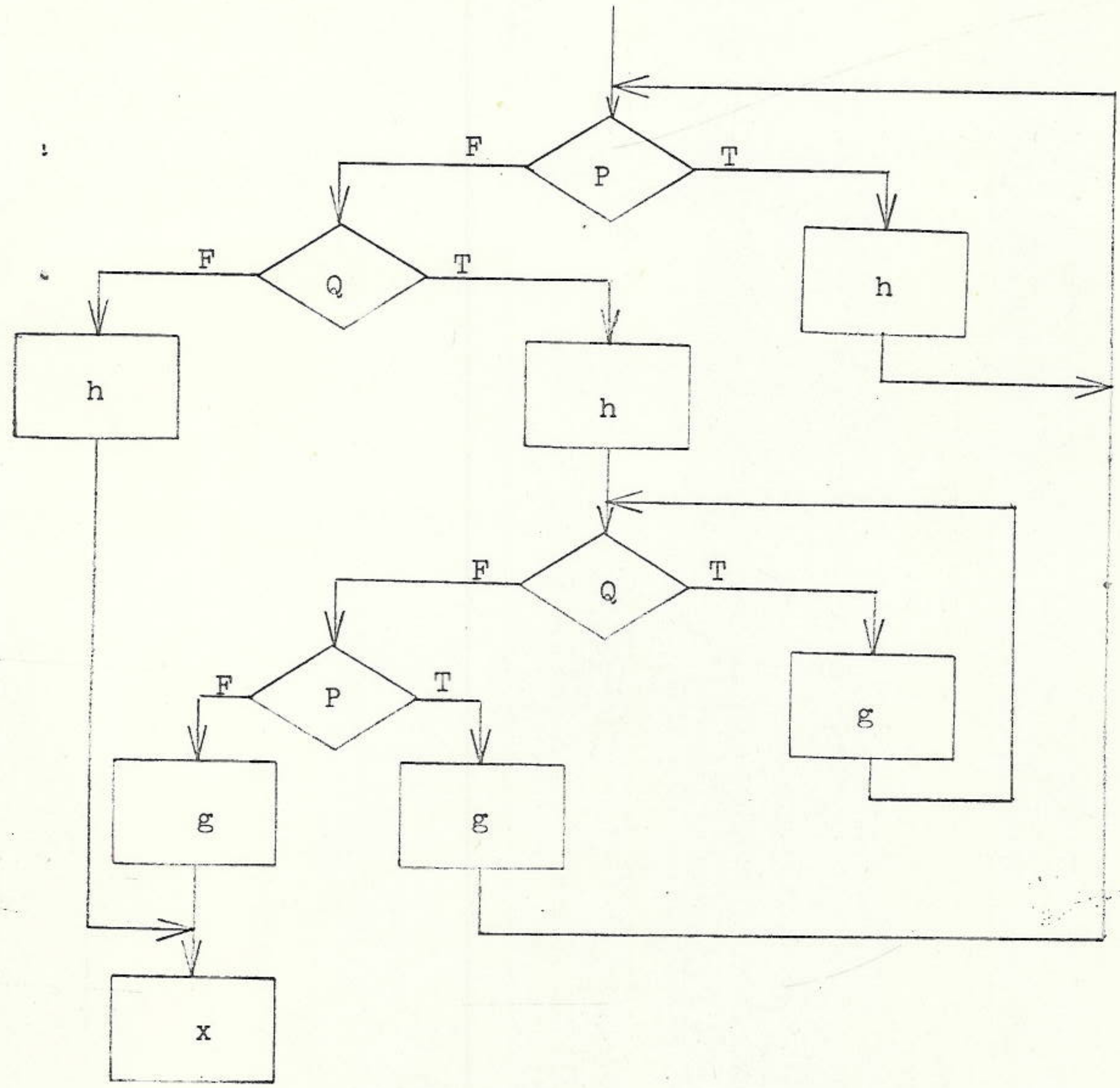

Fig. 6 Flowchart from Ashcroft and Manna [8].

Where test P means "is 'α' the leftmost letter in tail"; test Q means "is 'β' the leftmost letter in tail"; operation g means "erase the leftmost letter in tail and add 'g' on the right of head"; and operation h means "erase the leftmost letter in tail and add 'h' on the right of head".

References

1. Dijkstra, E.W. 1972. Notes on structured programming. Structured Programming, pp. 1-82. Academic Press, London.

2. Wise, D.S.; Friedman, D.P.; Shapiro, S.C.; and Wand, M., Computer Science Department, Boolean-valued loops, Technical Report No. 21, June, 1975

3. McCarthy, J., et al. 1962. LISP 1.5 Programmer's Manual. MIT Press, Cambridge, Mass.

4. van Wijngaarden, A.; Mailloux, B.J.; Peck, J.F.L.; and Koster, C.H.A. 1969. Report of the Algorithmic Language ALGOL 68. ACM, New York.

5. Wulf, W.A.; Russell, D.B.; and Habermann, A.N. December 1971. BLISS: a language for systems programming. CACM 14, 12:780-90.

6. Nassi, I., and Shneiderman, B. August 1973. Flowchart techniques for structured programming. SIGPLAN Notices 8, 8:12-26.

7. Peterson, W.W.; Kasami, T.; and Tokura, N. August 1973. On the capabilities of while, repeat and exit statements. CACM 16, 8:503-12.

8. Ashcroft, E., and Manna, Z. 1972. The translation of 'go to' programs to 'while' programs. Information Processing 71, pp. 250-5. North-Holland Publishing Co.

Typed by Christopher Charles